DBSMON

DataBase Monitoring

FERMILAB – CD - CEPA

Lee Lueking

Yuyi Guo, Jim Kowalkoski, Rodolfo Pellizzoni

Steve White, Eric Wicklund, Margherita Vittone-Wiersma

Table of Content

1. General overview

Monitoring the database operation is a general need for all experiments and a framework and tools are needed to enable this. In the summer of 2002, CDF began building a system that would collect udp messages sent from clients accessing their database. In the Fall of 2002, it was recognized that many of the tools being developed for CDF could be shared with D0, and other experiments, and a general framework for monitoring, archiving, and presentation was discussed. The DBSMON project was established to collect and build the tools necessary to perform the monitoring and systems have been put in place to automate the process. Significant effort has also been devoted to determining the kinds of information useful to be monitored, and to collect an historical summary which can be used to diagnose problems and monitor trends as the usage patterns evolve over time.

2. Description of the project

The main idea behind the project was to provide a way to monitor database access usage showing trends,stability and possibly help detecting problems for processes that are using the database in an unusual way.
The main effort was to find first commonality between the experiments , D0 and CDF in order to produce a monitoring tool as general as possible which could be reasonably extended to other future candidates. The focus was to identify useful information such as:

a) Top users , top applications, top tables accessed

b) Duration information for both queries and connections

c) Number of connections

etc.

A few decisions needed to be made:

a) Which database to use to store the information
b) Which language to use to write the code
c) Which charting and plotting tools were available

The decision was made to store the monitoring information in a MYSQL database : the simplicity of the schema and ease of administartion led us to choose the freeware over Oracle.
Java was chosen as the code language,easily portable, platform independent and easily interfaced to mysql .

The choice of plotting tools was guided by the nature of data to be represented and by the need of a user friendly display: bar charts, line charts, histograms. Details will expand in a following section.

The monitoring is provided through several jobs which run on a daily basis (once a day for CDF and on a 4 hour interval for D0) to produce plots available for the user through WEB access; the plots are also archived for historical usage.

Besides daily plots , summary plots are also provided which represents, for example, a month worth of data. Please refer to the appropriate section below.

3. General Features

One of the main tasks of the project was to find similar features in data coming from different experiments in order to provide a general framework that could be used in future experiments.
Since the purpose is to monitor database usage, the schema was designed to have as many tables as possible with similar fields to represent conceptually similar data:

- Timestamps
- Connections durations
- Tables most accessed
- Top users accessing the databases

All the previous item showed to be basically common .

CDF and D0 each have their own MYSQL servers and databases on separate machines.The mysql tables resides accordingly on different nodes.

3.1 Tables description

The base table that is used as originator of all the other information is the"Job" table;

Field	Type	Null	Key	Default	Extra
jobId	varchar(255)		PRI		
userName	varchar(255)		MUL		
nodeName	varchar(255)		MUL		
domainName	varchar(255)		MUL		
startTime	int(11)		MUL	0	
jobType	varchar(255)		MUL		
version	varchar(255)		MUL		
application	varchar(255)		MUL		

Describe fields here…

The jobId is the primary key that is used to join the other tables to retrieve the information to be plotted.

3.2 Plot generation and code description

The generation of plots has several steps:

- Java applications run in a timely fashion using cron jobs. The plots representing the information from the mysql db are saved on disk.
- Shell scripts invoking Python code move the plots in an established area available to the user from WEB access. Thumbnails are also created for easier navigation of the plots and they always point the user to the most up-to-date page; historical data is also available from an "Archive" link.

3.2.1 Plotting Tools

Plots are generated by Java applications which use two basic packages: JFREECHART and JAIDA.

To represent dbsmon data we needed charting and plotting tools to easily interpret the results: depending on the type of variables , we used :

- bar charts, line charts, scatter plots, time-series based plots
- histogramming

To achieve that we had to use two packages since Jfreechart did not provided the histogramming features.

JFREECHART is a free Java class library for generating charts, which can be downloaded from the web at http://www.jfree.org/jfreechart/ . The web site provides a general overview and useful demos; the full documentation needs to be purchased.

JAIDA was used to produce histograms ; this java package is also a freeware obtainable from http://java.freehep.org/jaida/index.html and it is part of the FreeHEP library The package provides a set of classes which are a java implementation of abstract interfaces or, more precisely for, AIDA - Abstract Interfaces for Data Analysis.

The histograms' content is then saved as datasets, basically arrays and then passed to Jfreechart charting and plotting methods.

3.2.2 Code Structure

DBSMON_GEN is the package that comprises all the source code to generate the dbsmon plots, from the java applications to the web pages; it is stored in cdcvs .
When checking out the product ,a directory structure will be automatically built. It has a top level GNUmakefile to make java executables. The basic directory structure follows:

```
./src
./src/cdf
./src/d0
./scripts
./test
./classes/
./classes/cdf
./classes/d0
./ups
./www
./www/cdf
./www/cdf/Rls/html
./www/cdf/Rls
./www/d0
./www/d0/Sam/html
./www/d0/Sam
./www/d0/CalibFarm/html
./www/d0/CalibFarm
./www/d0/CalibUser/html
./www/d0/CalibUser
```

The src directories contains the java sources; the built classes are found in the classes subdirectories.
The script directory has python scripts for web and plot generation configuration, as well as DDL’s for cdf base tables and cdf summary tables.

The www directory is the main tree structure for the web display and all the html’s for publication.

A basic applications is simply structured

- connection to database
- data retrieval
- dataset creation
- chart definition
- plot creation

All the java applications are individual classes which however make all use of a base class called DbAccess.java which provides methods to connect to the mysql databases; it also has few other simple utility methods . To be as general as possible connection parameters are read from a file, for example, for CDF:

```
------------------------------------------------------------------------------------------
// Database Access Config File //
// Arguments read:
//
// -s server name
// -d database
// -u username
// -p passwd

-s fcdfcaf057  -d dbsmon  -u cdf_reader -p reader

------------------------------------------------------------------------------------------
```

The following is the list of the currently available sources for both CDF and D0; the plots produced cover data for a 24-hour interval.

------------------------------------   CDF --

Name	Action
AllTables.java	Bar chart showing the top tables used
ConCountsFarms.java	Bar chart showing number of connections for FARM jobs
ConCountsNoFarms.java	Bar chart showing number of connections for NON-FARM jobs
ConDurFarms.java	Histogram showing connections duration for FARM jobs
ConDurNoFarms.java	Histogram showing connections duration for FARM jobs
ConJobFarms.java	Histogram showing number of connections per job for FARM jobs
ConJobNoFarms.java	Histogram showing number of connections per job for NON-FARM jobs
CpuHistory.java	Line plot showing top CPU users
CpuHistoryDaily.java	Line plot showing top CPU users
QueriesPerJob.java	Histogram showing number of queries
QueryDuration.java	Histogram showing durations of queries

QueryNumTime.java	Bar chart showing number of queries per hour
RecentUsers java	
TblsPerJob.java	Histogram showing table access counts
TopTables.java	Bar chart showing mostly accessed tables
TopUsers.java	Bar chart showing top users accessing the database
Versions.java	Bar chart showing different software releases in use

--- D0 ---

D0JobSrv.java	Bar chart showing number of connections for all the servers.
D0LatDurSrv.java	Histogram showing for each server a latency duration.
D0LatencySrv.java	Scatter plot for latency distribution counts
D0QryDurSrv.java	Histogram showing for each server queries duration.
D0QrySrv.java	Bar chart showing number of queries for all the servers.
D0QueryCnt.java	Bar chart showing number of queries for the day
D0ReqQrySrv.java	Line plot comparing number of requests and queries
D0ReqQrySrvMem.java	Line plot comparing number of requests and queries for cplus plus servers.
D0ReqSrv.java	Bar chart showing number of request per hour for the day
D0TopTables.java	Top tables accessed for all the servers.
D0TopTablesSrv.java	Top tables accessed for a specific server.
D0TopUsers.java	Top users for all the servers
D0TopUsersSrv.java	Top users for for a specific server.

------------------------------------- D0 SAM specific ---

SamQryCntsSrv.java	
SamQrySrv.java	
SamTopTables.java	
SamTopUsers.java	

The whole process of creating the plots is comprised of the steps listed below; multiple cron jobs run every night for CDF and D0; for D0 only, they also run at intervals of 4 hours.

- A cron job invokes a shell script to tun the java applications which create the plots

- A shell script invokes a python script which in turns reads a python configuration file.
- The configuration has a list of the names of the files, one for each plots; thumbnails are made to link to the most recent version of the plots.
- The plots are located in a designated area.

Archival information is also available, plots and log files, for now available as a list of files names.

3.2.3 Summary tables and plots

In a similar fashion as per the daily plots, monthly summary plots are also produced. The idea is to observe trends or special effects on the database access over a longer interval and summarize them in a single plot; summary tables exist and data is stored one a day using separate methods for D0 and CDF; for CDF java applications read the data from the dbsmon tables and store it in the summary tables; for D0 the equivalent is done by shell and sql scripts. Generation of plots then follows the same procedure as the one for the daily plots.

Access to the web pages for the summary plots is done from the individual dbsmon pages.

4. CDF Features

CDF database monitoring has client-server architecture. The clients are cdf offline jobs which using Error Logger package to generate and pack their database activity records. The clients send UDP package to the central collecting server called RemoteLogServer that again uses Error Logger package to unpack messages. In addition to the central server, Clients can also send records to a local log file that can be configured in the TCL files as well as turn on the debugging. The later two functionalities were heavily used before the database software got stabled.

4.1 Client and Server Model

Figure 1. shows the client-server structure used to transfer monitoring data from the client code to the database via the server.

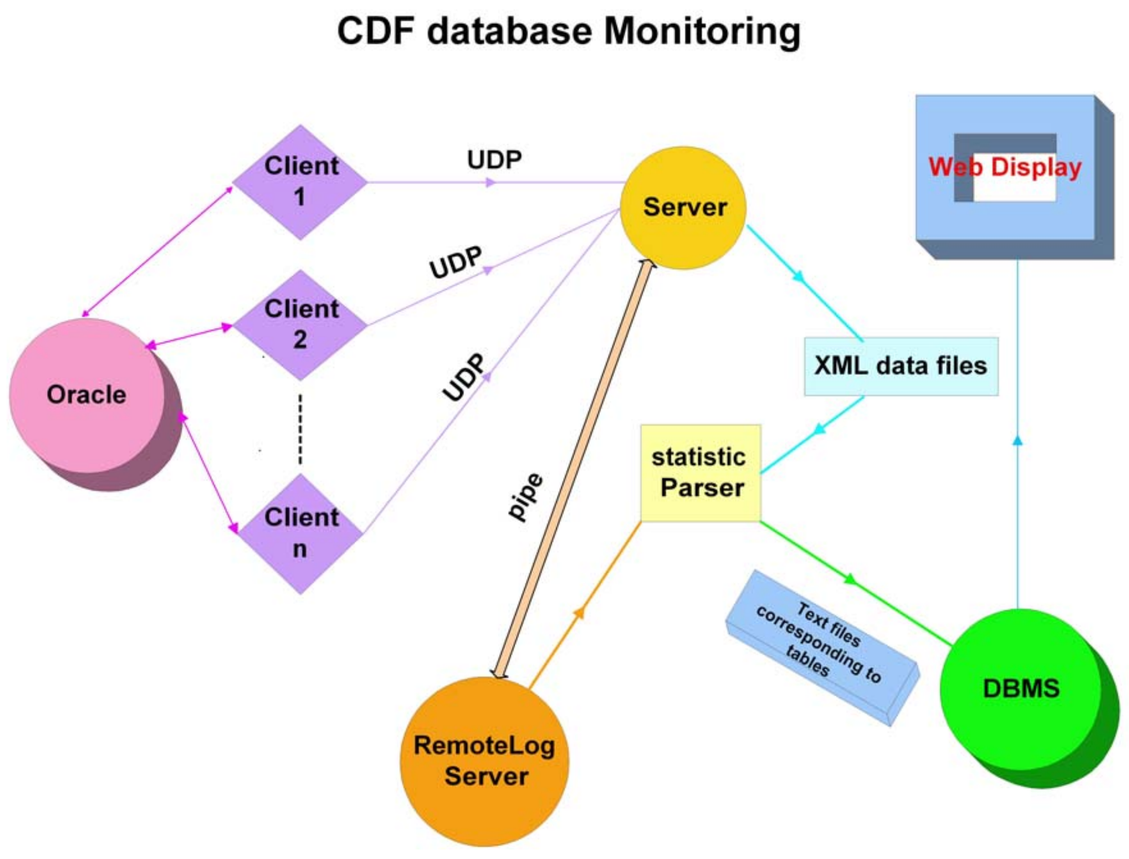


Figure 1. Client-Server Structure

We define client messages into different catalogues and severities according to the errorlogger formats: Severity Level, MessageID , Messages, Event Num, Run Num, Date & Time.

-- ELinfo, DBCON_J Start/Stop, sequence number , job id, node name, username, job type, job name, version.
-- ELinfo, DBCON_A, New, sequence number , job id, connection string, db id.
-- ELinfo, DBCON_A, Release, sequence number, Job id, flag, connection string, Duration.
-- ELinfo, DBCON_A, Reconnect, sequence number, job id, connection string.
-- ELinfo, DBCON_O, OTLnew/OTLrelease, sequence number , Job id, connection string.
-- ELerror, ConFail, sequence, job id, connection string, retry number.
-- ELsuccess, DBACC_G, API-keyed, sequence number, job id, duration, table name, db id.
-- ELsuccess, DBACC_G, API-unkeyed, sequence number, job id, table name, db id.
-- Elincidental, DBSQL_O, Sql, sequence number, job id, query string.
-- Elincidental, DEBUG ….
All these messages are at low level of cdfsoft in each application. They are defined in DBManager package and implemented in FrameMods, DBManager and DBObjects package. Users don't do anything for it and they are set at ELsuccess level by default. But, this also gives the clients abilities to tune the levels of message to be recorded. For additional details refer to:
http://cdfkits.fnal.gov/CdfCode/source/DBManager/doc/informalProposal.txt .

The central server is implemented in RemoteLogServer package. The package includes the following files and directories:

- GNUmakefile standard package-level makefile
- RemoteLogServer/ contains all package header files

  - RemoteLogServer/link_RemoteLogServer.mk standard link file; RemoteLogServer, on the client side, depends exclusively on ErrorLogger package .
  - RemoteLogServer/ClientList.hh This file implements a list of clients. It is used by UDPserver to maintain a record of all connected clients and packets received. See network protocol for more information.
  - RemoteLogServer/PacketList.hh A list of packets sent by a client. This is used by ClientList.
  - RemoteLogServer/sock_interface.hh This is a workaround for compiling with KCC under linux. It contains declarations of external C procedures defined in sock_interface.c. These functions actually do nothing more than calling the equivalent unix system

    procedures, but this way we dont have to include the standards header files in C++ code. Including them would in fact cause en error because of the presence in file bits/in.h of a struct containing a member with the same name of the struct itself. This is legal C code but not C++ (*check if true*), anyway many compilers get it right, including g++. Unfortunatly KCC does not.
    - RemoteLogServer/UDPclient.hh The client code exhibits a send(int num, const char* message) member which is used to send logs over the network. See client configuration for more detail on the inner functioning of the class.
    - RemoteLogServer/UDPsender.hh This is a subclass of ELsender. It is used together with an ELdestination to log messages via the ErrorLogger facility.
    - RemoteLogServer/UDPserver.hh The server class. See network protocol and server configuration for more details.

- src/ contains sources for client portion of the code.

    - src/.refresh
    - src/GNUmakefile this builds library libRemoteLogServer
    - src/sock_interface
    - src/UDPclient.cc
    - src/UDPsender.cc

- srcserver/ contains sources for client portion of the code.

    - srcserver/.refresh
    - srcserver/GNUmakefile this builds library libRemoteLogServerServ
    - srcserver/ClientList.cc
    - srcserver/PacketList.cc
    - srcserver/sock_interface.c
    - srcserver/UDPserver.cc

- exe/ sources for building the server application

    - exe/GNUmakefile
    - exe/server.cc server program. See server configuration for more details.
    - exe/RemoteLogServer deamon code. See deamon configuration for more details.
    - exe/RemoteLogServer.config deamon configuration file.
    - exe/server.config server example configuration file.

- test/ test programs

    - test/.binclean
    - test/GNUmakefile builds all tbin and test trytime, tryUDP and trysend.
    - test/trytime.cc first integration test program. Since the server unfortunately depends on a certain implementation of type time_t and long int, we need to check them.
    - test/tryUDP second integration test program. It tests the UDP protocol.
    - test/trysend.cc third integration test program. It tests the server/client protocol.
    - test/UDPColl.hh include file for one of the three client-side test program. This one, taken from ELColl in package ErrorLogger, produces a small output of 6 logs.
    - test/UDPColl.cc
    - test/UDPTestClient.cc used to run a stress test on server. It continually logs messages to the server at maximum speed.
    - test/UDPTestRealClient.cc used to run a simulation of a real cdfoffline job. See test suite.
    - test/runmultiple.cc used to run multiple instances of UDPTestRealClient.
    - test/UDPTestRealServer.cc a simple server, which logs messages to the screen
    - test/UDPTestDataServer.cc another server, which expects to see messages coming from database accesses and dump them of files based on jobid.

- scripts/ contains various scripts

    - scripts/GNUmakefile builds script installserver
    - scripts/installserver install the server as a deamon. You must have administrator privileges to run this shell script.
    - scripts/RemoteLogServer deamon script to be installed in /etc/rc.d/. See deamon configuration.
    - scripts/myzip zip the archive files .
    - scripts/UDPserver.cxx root file for data analysis

4.1.1 Server Network Protocol

UDP was choosed to implement the client-server protocol because it allows us to use a single connectionless socket on the server (while tcp would force us to use multiple sockets) and need no connection on the client side (we really don't want the client to worry about the server status once it connected).

4.1.2 Client Packaging

When the client is created, it first attempt to connect to a server (see Client configuration). To do so it sends a "ping" packet over the network at a predefined port (port 1), using this format:

PID of client (pid_t, 4 bytes)	Version number (int, 4 bytes)

The version number is meant to provide an opportunity to upgrade the protocol while keeping backward compatibility. Only version 1 is currently implemented. The client then wait for the server response to be sent. The server answer by sending back the same packet. The client won't wait forever though; after MAXWAIT seconds it will try sending another packet to the server an so on for MAXATTEMPTS time, after that it will consider the server as unreachable and, if provided with more servers, it will try another one until the end of the server list. If the client is ultimatly unable to connect to any server, it won't send any more packets and report false if connected() is called.

The client receives messages through the send(int num, const char* message), after they have been converted into a char string by an ELdestination instance. The client then packs them into longer string to be send to the server via port 2, using following format (example for packing of two messages):

PID of client	Packet number (int, 4 bytes)	Number of bytes (long, 4 bytes)	Message (char string)	Number of bytes	Message

Where Packet number is a progressive number starting from 1 and Number of bytes is equal to the length of the message (no line feed is added). After sending a message, the client expects no answer from the server nor will it send a message before the end of the job.

4.1.3 Server unpackaging

The server opens two port (port 1 and port 2), then waits until a packet arrives. Due to the unreliable nature of UDP, some controls have to be done in order to ensure correctness of data inflow. When a client first connects to port 1, the server checks if the client already sent packets and if not it records the client in an ad-hoc list (ClientList), together with its version number and a packets list. Each time a packet arrives on port 2, the server checks that:

- the client has already connected on port 1
- the packets are arriving in the rigth order

If anything is wrong, the server tries to correct it in the best way it can, including:

- resorting out of order packages

- discarding duplicated packages
- account for lost packets

In order not to consume all system memory, clients wich have been founded to be inactive for the longest time are periodically deleted. Note that this means keeping an ordered list of packet arrivals with an LRA (least recently accessed) algorithm. ClientList does a good job in doing that.

Any correctly arrived packets is then send to the ErrorLogger facilities for string unpacking.

4.1.4 Client Configuration

The client code (class UDPclient) is never directly call. Instead, a call to one of the makeDestination functions is performed:

```
bool makeDestination(std::string serverName,unsigned short
portNum,ELdestination*& dest);


bool makeDestination(std::string serverName,unsigned short portNum,int
bufferSize,ELdestination*& dest);


bool makeDestination(const std::vector<std::string>& serverName,const
std::vector<unsigned short>& portNum,ELdestination*& dest);
bool makeDestination(const std::vector<std::string>& serverName,const
std::vector<unsigned short>& portNum,int bufferSize,ELdestination*& dest);
```

These functions create a new ELdestination object (actually an instance of a subclass of ELdestination) and pass it back through pointer dest. This object can be actually attached to an ErrorLog instance via ELadministrator, so that messages logged through the ErrorLog flow to an UDPclient object (see this code for example). Please note that you must make sure to delete dest. The deletion of the UDPclient object is taken care automatically by the ELsender class, so you don't have to worry about it.

You must pass the following info to makeDestination:

- either a server name or a list of servers. Each one can be a host name or an address in the four dot notation.
- a corresponding server port or list of ports
- an option buffer length, which is the maximum length in bytes of packets sent by the client

Two important defines are located in UDPclient.cc:

- MAXWAIT sets the maximum time in second the client is willing to wait for a server ping response
- MAXTRY sets the maximum number of times the client is willing to try connecting to the server

Note that if all servers are down, the client job will have to wait for MAXWAIT*MAXTRY*Number of servers seconds.

4.1.5 Server Configuration

The server network code is implemented by class UDPserver. /exe/server.cc contains the code for message logging management.

The server will log different message types to appropriate message log files, which are named after the MySQL tables. To match each message types against each tables, see files MySQL tables.txt and informalProposal.txt. Class UDPserver will also create three log files to record network related events.

To start the server, call it by passing on the command line the name of a configuration file. In the file, blank lines and lines beginning with # are ignored; all other lines must contain an parameter name followed by its value. The valid parameters are:

- num_messages :   the number of message to be logged to every message log file. After this number is exceeded, the server will create a new file (each file carries a progressive number in its name)
- num_files :   the number of message log files of each types to be created. After num_files are created, the server will start overwriting the first one
- port: the server port number. This will become port 1; port+1 will become port 2. Please note that this parameter is mandatory; all others are optional
- logLevel: the log level for network information events. See file UDPserver.hh for the available levels and their meaning.
- logFile: the network log file for displaying informational events.
- ErrorFile: the network log file for displaying error events.
- interFile: the network log file for storing packets interarrival informations. (*we need to be able to disable this feature*)
- idleError : the minimum idle processor time, in percentage, on the server machine. If idle times drop below this percenetage, a MAILERROR will be issued. Set to zero to disable feature
- meanMail: the maximum acceptable mean packets lost by second. If the mean raises above this value, a MAILERROR will be issued. Set to zero to disable feature
- mail :   list of white-space separated mail addresses to which errors of MAILERROR level will be sent.
- maxLines :   maximum number of lines in each network log file. The server will then create a new network log file with increased progressive number.

- maxLogFiles: number of network log files, after which the server will start overwriting
- binSec: the time resolution (bin) used to output information to the network log files
- numBin: the number of preceding time intervals used to compute means (ex: if you set binSec to 120 and numBin to 5, than you get a new mean value every 2 minutes and that mean is computed over the last 10 minutes)

See also file server.config for an example and UDPserver.hh for default values.

If you run the child be forking it as a new process, you can also specify a file descriptor number after the configuration file. In this case the server will send a file name to that descriptor each time it closes a message log file. The deamon takes advantage of this behaviour.

4.1.6 Daemon Configuration

Daemon Configuration

A daemon program (RemoteLogServer) is provided to conveniently run the server.

The daemon takes care, among other things, of:

- restarting the server in case of crashes
- logging all server messages via the syslog system facility
- reading message log files and load them into the MySQL database.

After you build the packet's bin, you can install the daemon by running file installserver. You must have root privileges to do that. The script will:

- install a standard script in /etc/rc.d/
- install RemoteLogServer to /usr/bin/ and its associated configuration file, RemoteLogServer.config, to /etc/
- create a new directory /usr/RemoteLogServer and install the server there, toghether with its configuration file and a script (mysqlloader) for loading data into a MySQL database

You can also install everything manually by moving files to the right directories. In that case please check file RemoteLogServer.config. It contains two lines:

- the first one is the directory where you want to put the server
- the second one is the path to the MySQL script; it can be relative to the first directory. If you want to disable writing to the MySQL database, set this line to none

You can easily change the database logging system by writing a new script and update RemoteLogServer.config. The deamon will pass two command line arguments to the script:

- the message file name to load
- the table name

this should enable to keep the script really simple and small.

4.2 Test suite

Class UDPserver is designed to provide full support for testing and logging data on server behaviour. The data can be further analyzed using statistical analysis systems such as root. The package also provide client programs for simulating real cdf offline jobs.

4.2.1 Test set up and result

The server outputs data to three network log files based on the desidered logLevel. Setting logLevel to EVERYTHING will produce an output of all received packets, and it's generally not recommended. The default value (REPORT) will log all information needed for data analysis. Just start up root and load file UDPserver.cxx. CINT (the root interpreter) does not do a good job when it comes to the STL, so it's better to compile UDPserver.cxx as a dynamic library (use the root command .L UDPserver.cxx+, assuming you have the file in your working directory). UDPserver.cxx provides five methods, all of them return a pointer to a TH1D object (a one dimensional histogram of doubles). All five methods take the network log file name as their first argument. All but the last also take a double as their second argument, it should be equal to the number of seconds set in binSec, or the same time in minutes or hours if you prefer.

- MeanPackets: extracts information about the mean number of valid packets arrived per second
- MeanIdle: mean processor idle time on the server machine, in percentage
- MeanError: mean lost packets per second
- MeanMemory: memory occupation in Mbytes. To produce a suitable input file for this method, you can use top -d num_seconds -p server_pid -b > output.file and then grep server output.file > output (*a built-in functionality would be preferred; anyway the maximum memory occupation is essentially fixed through the define MAXNUMCLIENTS and MAXDIM in ClientList.hh and PacketList.hh*).
- Interarrival: interarrival time between packets, bin is 1 millisecond

4.2.2 Client Job Simulation

To use client simulation programs, use server UDPTestRealServer. This server will output all incoming messages to the standard output and do nothing else, apart from producing network log files. UDPTestClient can be used to generate large amount of messages as fast as possible. This act essentially as a stress test and is indeed not very usuful. UDPTestRealClient instead simulates a cdf offline job by sending messages in a sensible pattern, following this pseudocode:

```
while(job running for less than job_length_sec seconds) {

   //simulate new run begin where db accesses occur

   for(int i=0;i<n_short_accesses;i++) {

      generate two or three messages

      sleep(short_length_msec milliseconds)

   }

   for(int i=0;i<n_long_accesses;i++) {

      generate two or three messages

      sleep(short_long_ssec seconds)

   }

   sleep(run_length_sec seconds)

}
```

The preceding parameters, plus more, can be passed as command line argument to the programs. Running UDPTestRealServer without arguments will give the complete list. Note however that the time parameters are randomly variated.

Program runmultiple is used to run mutiple instance of UDPTestRealClient with the same parameter list. It accepts the same arguments as UDPTestRealClient plus three more. The client number will initially grow slowly, based on the wait_time argument (runmultiple will sleep for wait_time seconds each time it forks 10 children) and then will be kept to n_clients. The simulation will last for total_length_time seconds. \

4.2.3 Test set up and result

Using this facility some load tests have been run. Three use cases have been isolated:

- reconstruction job run on a farm
- analysis job
- misbehaving job

The jobs have been simulated with the following parameters:

	Number of long accesses	Number of short accesses	Run length	Long access time	Short access time	Job length	Number of jobs
Reconstruction job	10	40	120 sec	30 sec	200 msec	1 h	200
Analysis job	7	28	30 sec	30 sec	200 msec	30 min	500
Misbehaving job	0	20	5 sec		200 msec	2 h	20

The simulation ran for 700 minutes.

The server generates an xml file whose size can be configured in the server configuration file. When the size reaches its maximum, the statistical parser will load it into a mysql database.

4.2.4 Database Tables

Dbsmon has common job level tables across all the experiments. The common tables (Job and JobEnd) describe when a job start/stop, its Id (nodename_pid_unixtime) which is unique accross all the jobs, username, node name, domain name, job type, version, application name. Cdf has its own tables too:

Low level tables:

- OTLcon: record when a connection happens at OTL level.
- OTLquery: record when a sql statement executed at OTL level.
- ConFail: record when a connection is failed and the number of time it tries.
- Sql: record the sql statement send to Oracle at OTL level.

Calibration API level tables:

- New: record when the first time connection established.
- Reconnect: record every time except the first time when a connection established.
- Release: record connection duration when a connection is released.
- APIkeyed: record query duration and table info when an Oracle table is actually accessed.
- APIunkey: record table info when it’s accessed from cache.

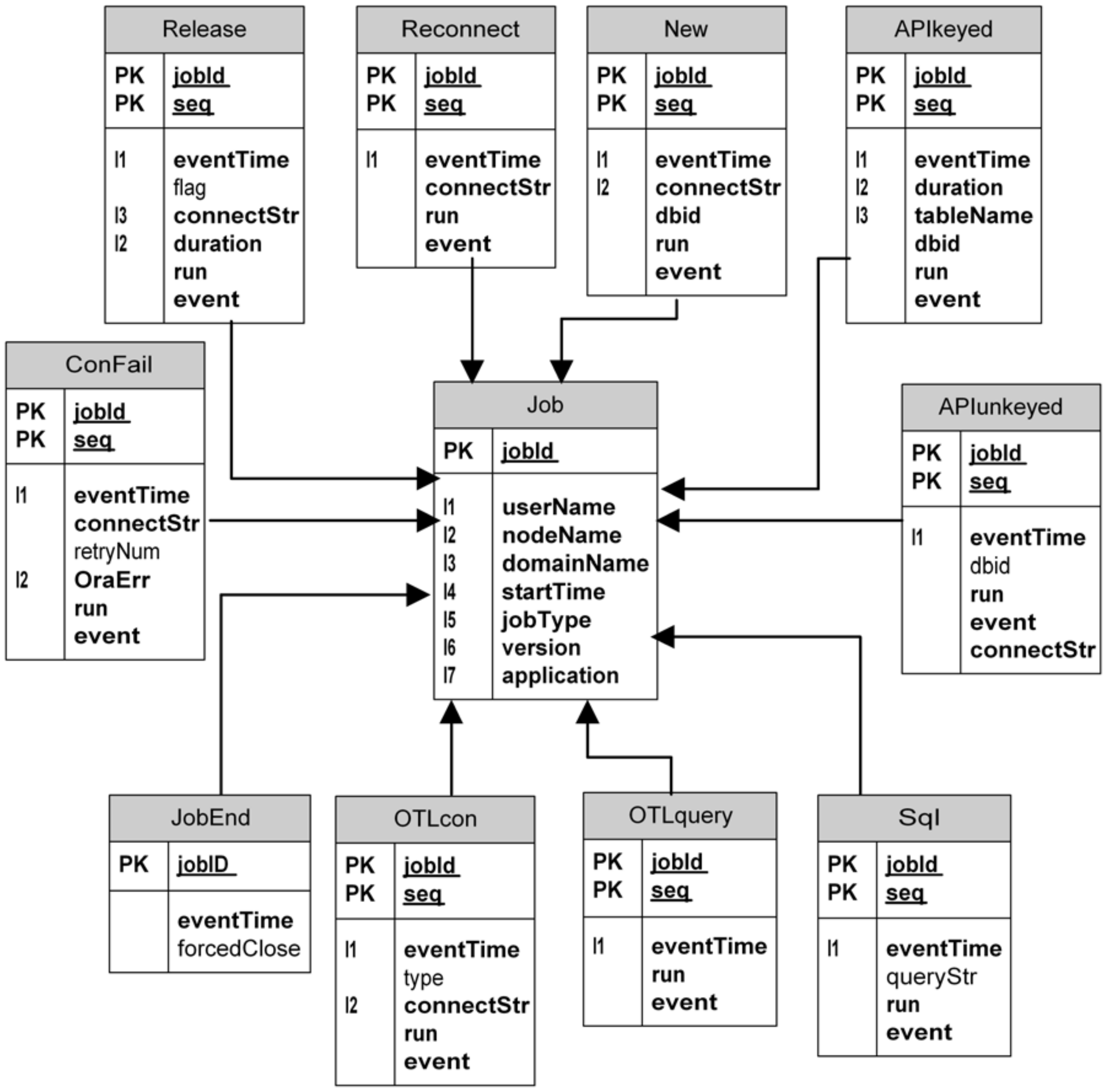


Figure 2. Schema for CDF

Working notes
Please refer to http://home.fnal.gov/~yuyi/dbsmon_cdf/dbsmon_cdf.html .

4.2.5 Database Cleanup

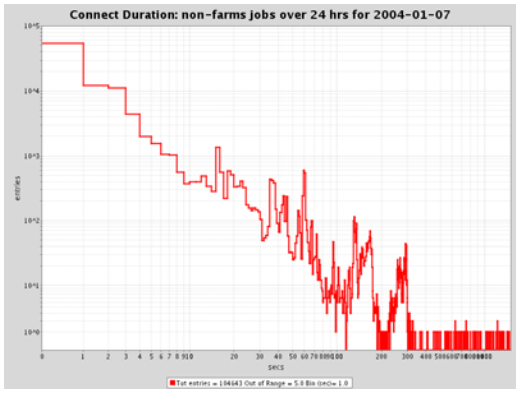
The database cleanup for cdf is done by a cdcvs package called dbsmon_maintain_cdf. The database is cleaned up every night by cron job. Tables are moved into a secondary database during the cleanup and old data is deleted from the secondary database, then we import data from the secondary db into the primary db. Because we are getting huge data volumes, each table has different data keeping rules according to their size and importance. Details can be found in dump directory of dbsmon_maintain_cdf package.

In addition to the cleanup, we have a watchdog script to restart the remotelogserver in case it can not be started by itself. And a makeArchive script to move the source data into an archive directory for waiting another process to move them into enstore

5. CDF PLOTS description

Connection Duration for database instance cdfrep01

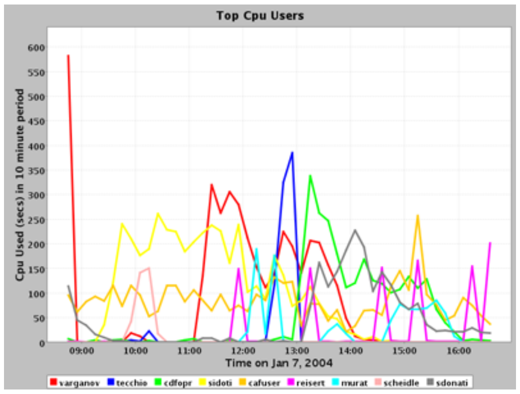
http://dbsmon.fnal.gov/cdf/Rls/html/CdfOfRead_CnDur.html



This plot shows the durations of connections to database instance cdfrep01, which is the database accessed by average users. One day of data is shown. A log-log scale has been chosen to show the long connection durations, which are of particular interest. One can click on the "Click for Data" link above to see the details of the long connection durations. The information for this plot is drawn from the table Release in MySQL database dbsmon.

CPU History for database instance cdfrep01

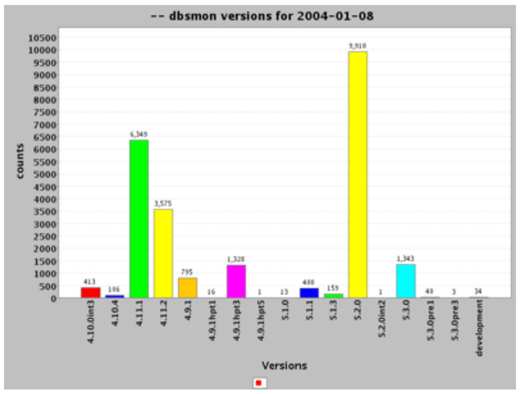
http://dbsmon.fnal.gov/cdf/Rls/html/All_CpuHistory.html



This plot shows the top users of CPU on database instance cdfrep01, which is the database accessed by average users. Measurements are made every ten minutes. If a particular user has an anomalously large usage of CPU, that shows up clearly in this plot. The information for this plot is drawn from the Oracle table LOGINTRACES which is filled by the Database Administrators.

CDF Software Versions

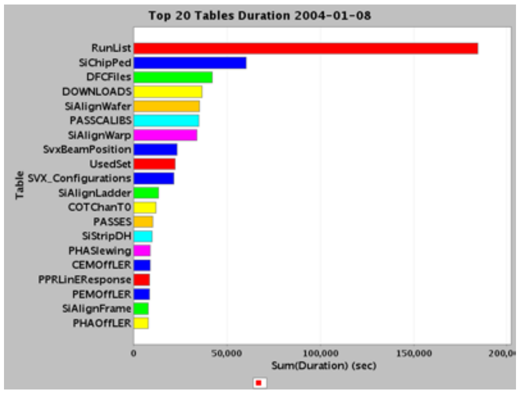
http://dbsmon.fnal.gov/cdf/Rls/html/All_Versions.html



This chart contains one entry for each job (or job section) run on CDF computers on the previous day. The jobs are broken out by CDF version, which shows clearly which versions are in use, and how heavily used each is. The information for this plot is drawn from the table Job in MySQL database dbsmon.

Top 20 Tables

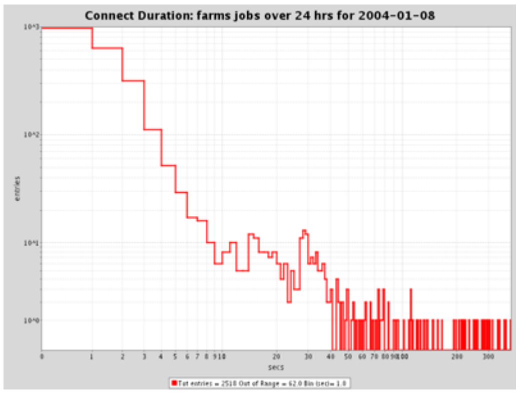
http://dbsmon.fnal.gov/cdf/Rls/html/All_TopTbls.html



The chart above shows the total elapsed time taken in each database table. A caveat: we are aware that certain tables, for example RUNCONFIGURATIONS, SET_RUN_MAPS and PARENTS do not appear in this chart. The information for this plot is drawn from the table APIkeyed in MySQL database dbsmon.

Connection Duration for database instance cdfofprd

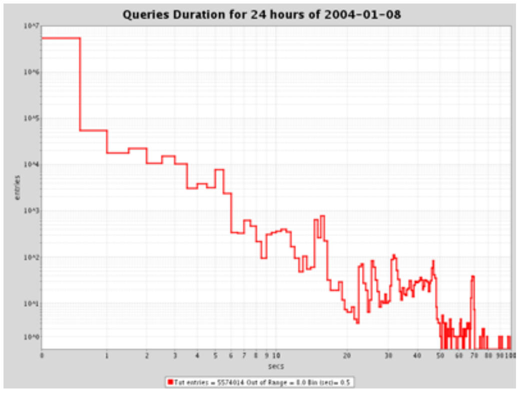
http://dbsmon.fnal.gov/cdf/Rls/html/CdfOfPrd_CnDur.html



This plot shows the durations of connections to database instance cdfofprd, which is the database accessed by the production farm. One day of data is shown. A log-log scale has been chosen to search for long connection durations, which are of particular interest. The information for this plot is drawn from the table Release in MySQL database dbsmon.

Query Duration

http://dbsmon.fnal.gov/cdf/Rls/html/All_QryDur.html

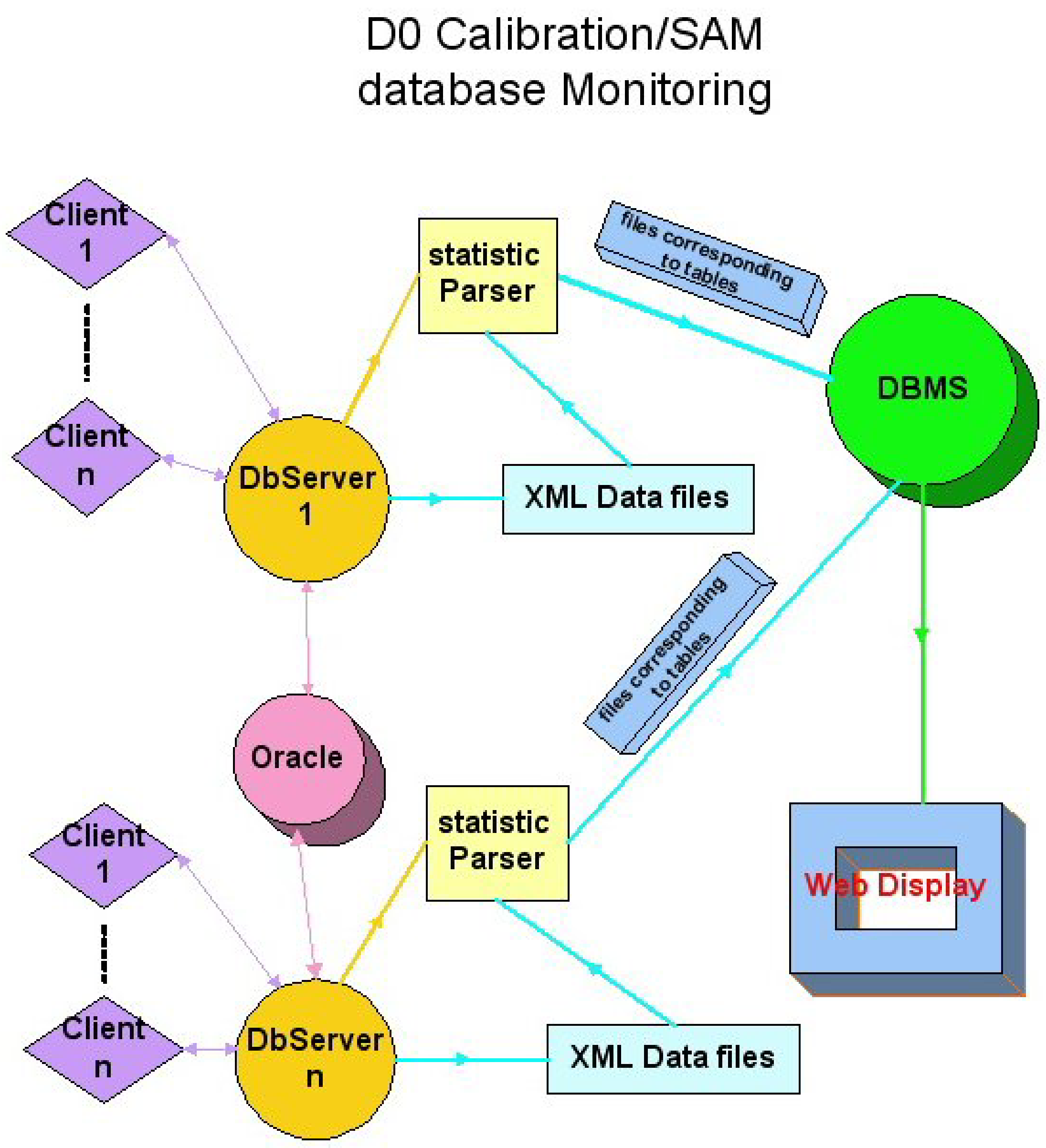


This plot shows the durations of queries made to database instances cdfrep01 and cdfofprd together. One day of data is shown. A log-log scale has been chosen to show the long query durations, which are of particular interest. The information for this plot is drawn from the table APIkeyed in MySQL database dbsmon.

6. D0 Features

Components - dbsmon_col

Dbsmon_col is a subset of the dbsmon product whose purpose it to receive a file of events and store them in a mysql database. It sits between a user built client application and a mysql database. Both CDF and D0 use this product but, in slightly different ways. A picture of D0's user of dbsmon is shown below.



The main component of dbsmon_col is statisticsParser.py (SP.py). SP.py is launched from the client application with a file of events. The events must be in a specific XML defined format. SP.py will parse this file into a set of text based tab delimited files where each file holds the events for one mysql table. If configured to do so, SP.py will load each of the files into mysql and rename the file when completed. The configuration parameters for SP.py are as follows:

statisticsParser.py -d directory [-s] [-l node,user,password,database] [-e experment] [-p readPrefix,savedPrefix,skip] [-w] [xmlFile xmlFile...]

Parses the xml data into separate tab delimited files. If the '-l' flag is provided, the files are loaded into a Mysql database and then deleted. If not provided, the files will neither be loaded nor deleted.

-d directory where statistics files exist, parsed files will be created loaded and then removed.

-s skip outputting an event_format file.

-l load the parsed data into the database.
node - node where database is located
user - user to log in as
password - password to use
database - database to set to
-e experment data is being created for. Defaults to d0

-p readprefix,savedprefix,skipLast readprefix is the file prefix to search for. All files with that prefix will be parsed. If the -l flag is set the prefix will be replaced with savedPrefix after the file is stored in the database. The skip can be 0 or 1 or a file name. If 1, the last file will be not be parsed. If 0 all files will be parsed.

-w cause a log file to be written

xmlFile 1..N - are the XML statistics files to parse. XML Files must be provided in the order they were created! Each file can include the path.

NOTE: either -stub must be supplied OR the xmlFile(s), If both are specified -p will be used.
The storage of events into the file is done in XML format with each event type fully described by meta data. This meta data will appear before the use of the event. Each user will be completely identified by meta data which will also appear in the file prior to its use in an event. The event itself will only contain the data unique to the event.

6.1 XML file description

Each XML event file will begin with the following:

<statistics app=”application” version =”v1_0” pid=”3242”>

- App – STRING filed for application name.
- Version – STRING field for ups version number
- Pid – INTEGER field for process id.

Each of the above relates to the dbserver application.

The XML event file will end with:

</statistics>

6.2 Describing the Identity of a User.

When a new user (client) is identified to the collecting application it creates unique event called “identitydesc”. In cases where the dbserver does not know who its clients are, this data will represent the dbserver itself. Its details are described below.

<identitydesc jid=”unique job Id” timestamp=1032835396>

- jid - STRING field defined as shortNodeName_pid_unixtime. This is the unique key which will relate all events to a specific user.
- Timestamp – INTEGER filed for when the event was created.

<node value=”shortNodeName”/>

- STRING field - example “d0mino” or “d0lxbld1”….

<domain value=”domainName”/>

- STRING field for the domain only. (fnal.gov or what ever)

<user value=”username”/>

- STRING filed for the user’s login name.

<pid value=”process identification”/>

- INTEGER field. This data is not expected to be stored in the database.

<app value=”application”/>

- STRING filed for client application’s name or server’s name if the server is not given the client name.

<argc value=”arguments”/>

- STRING field for arguments. This data is not expected to be stored in the database.

<jobtype value=”userDefined”/>

- STRING field that CDF needs. D0??

<version value=”v2_4”/>

- STRING filed for the ups version number of the client.

</identitydesc>

- Terminator of each identity description.

6.3 Event Meta Data.

Each event type must be fully described by its meta data. The meta data must be produced before the event is used. There are three sections to defining an event’s meta data. They are identifying the event, describing the categories the event fits in and identifying the data in an event.

Event meta data is identified by the following:

<ename=”eventName”>

- ISTRING field. Name of the event, also the name of the mysql table the event data will be stored in. The event name must be unique and consistently used across all instances of the running application.

A developer can assign events to “categories” and define what those categories are. An example of this is:

<category cat=”class” name=”Sql”/>

- Events will be tagged in the database with the category or categories the developer assigns. This allows events to be queried and grouped by those categories. Categories are not required but are recommended.

Each event must describe the data it will record. There are two types of data primitive and complex. Primitive is simply ints, chars, floats…. Complex is a collection of primitive data or other complex data. Complex data is still under discussion and will not be further detailed.

Primitive data is defined as follows:

<primitive type=”int” name=”dataname”/>

The types of data currently supported are string, double, float, int, date, and boolean.

</edesc>

- Terminator of each event description.

6.4 Recording events.

Notable events will be recorded in the XML file with an XML tag of “event”. The general format of an “event” is as follows.

<event ename=”eventName” jid=”jobId” seq=”sequenceNumber” tstamp=”timestamp”>

data<tab>data<tab>data…..

</event>

Each event requires an event header. This header uniquely identifies each event and relates it to its identityDesc.

- ename is the name of the event as described in the meta data.
- jid is the job id of the user who generated this event. It must have first been entered into the XML file in an identDesc.
- seq is a sequence number for this event, required when a client’s call creates multiple events within one second.
- tstamp is the unix time of when the event occurred in seconds.

The second line lists the data associated with each event. This data MUST be entered in the same order it was declared in the meta data. Each datum must be separated by a tab “\t”.

The third line is the terminator for each event.

Components – Schema

The schema for dbsmon has created in layers. There is a generic set of tables that must exist in all mysql databases. Then there are the tables that are specific to the experiment. D0 supports two addition layers of tables, one for all tables used by db_server_base and one for SAM. CDF has only one additional layer.

Fig. 3. shows the top layer consisting of all common tables is below.

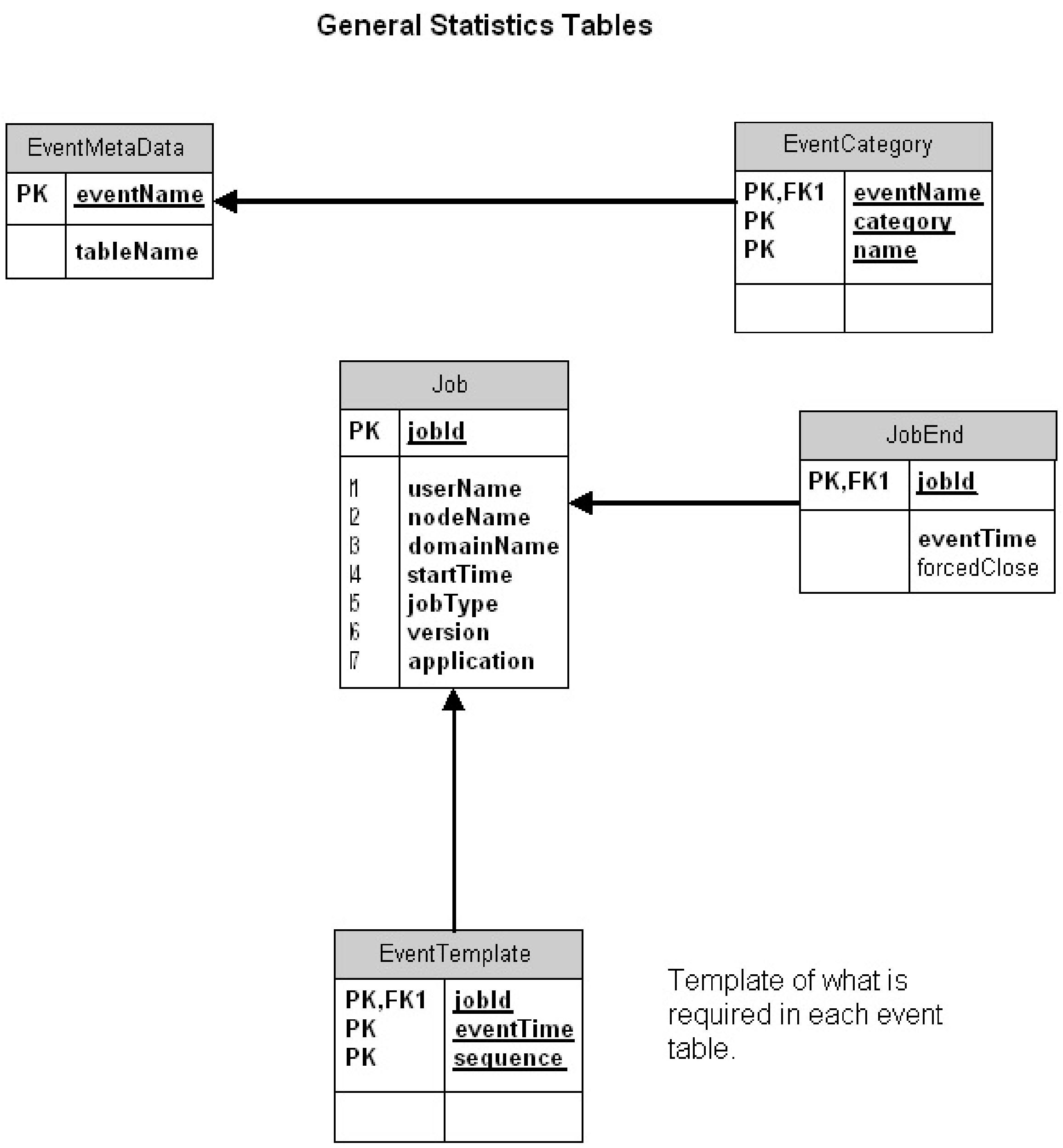


Figure 3. General Statistics Tables

These are the tables that are common to all dbsmon databases. A brief description of each table is provided:

- EventMetaData - Provided for use of applications outside of dbsmon, this table identifes the event and where it is stored. If used, the user application must insert this data.

- EventCategory – Provided for use of applications outside of dbsmon, this table identifies all categories the event belongs too. If user, the user application must insert this data.
- Job – Uniquely identifies the start of a job and details on who and where the job was started.
- JobEnd – Shows when a Job was terminated and if it was forced to be closed by some other application.

For D0, the next layer of tables is shown in Fig. 4. This layer is used by db_server_base.

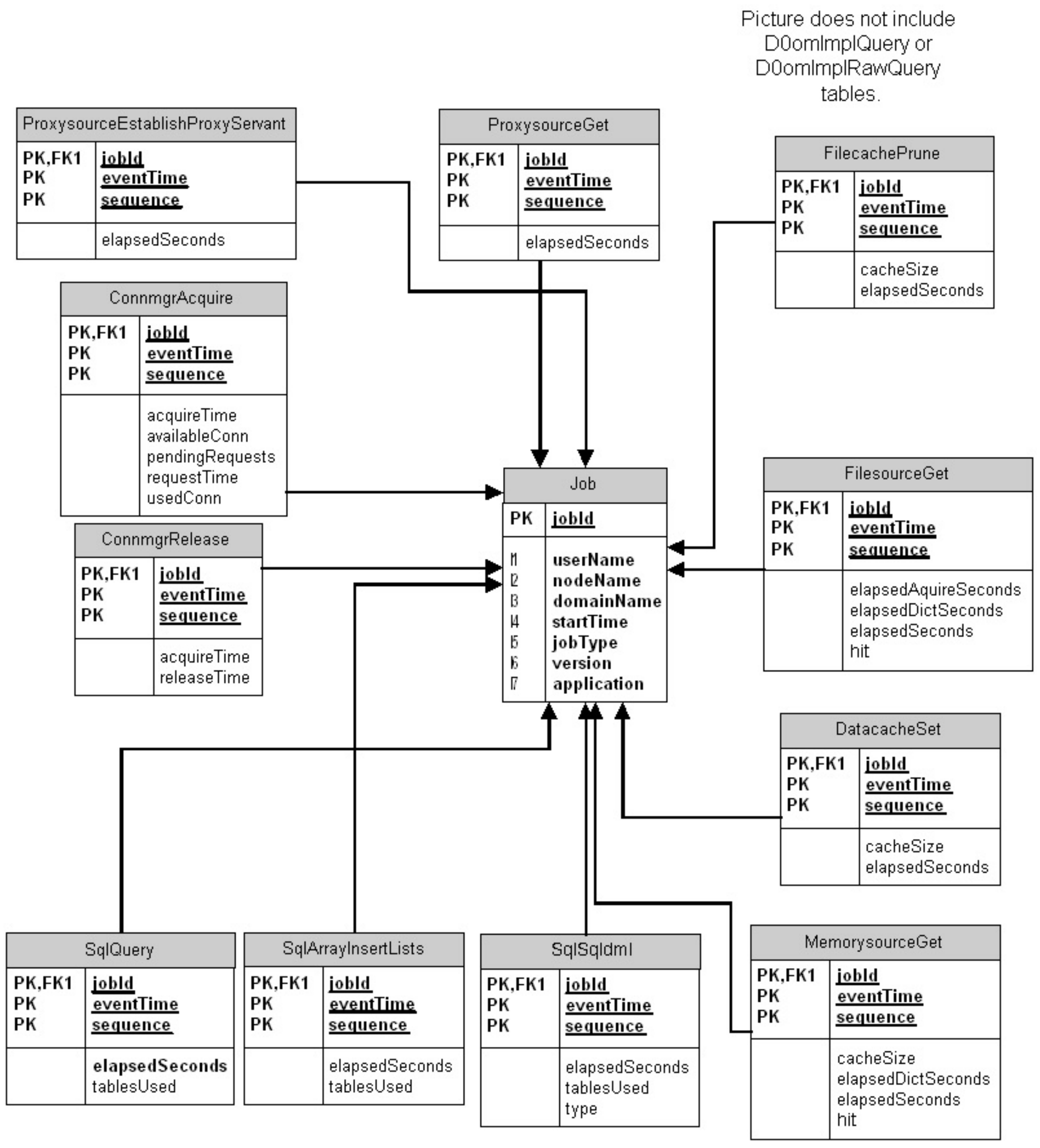

Picture does not include D0omImplQuery or D0omImplRawQuery tables.
ProxysourceEstablishProxyServant: PK,FK1 jobId; PK eventTime; PK sequence; elapsedSeconds
ProxysourceGet: PK,FK1 jobId; PK eventTime; PK sequence; elapsedSeconds
FilecachePrune: PK,FK1 jobId; PK eventTime; PK sequence; cacheSize; elapsedSeconds
ConnmgrAcquire: PK,FK1 jobId; PK eventTime; PK sequence; acquireTime; availableConn; pendingRequests; requestTime; usedConn
ConnmgrRelease: PK,FK1 jobId; PK eventTime; PK sequence; acquireTime; releaseTime
Job: PK jobId; I1 userName; I2 nodeName; I3 domainName; I4 startTime; I5 jobType; I6 version; I7 application
FilesourceGet: PK,FK1 jobId; PK eventTime; PK sequence; elapsedAquireSeconds; elapsedDictSeconds; elapsedSeconds; hit
DatacacheSet: PK,FK1 jobId; PK eventTime; PK sequence; cacheSize; elapsedSeconds
SqlQuery: PK,FK1 jobId; PK eventTime; PK sequence; elapsedSeconds; tablesUsed
SqlArrayInsertLists: PK,FK1 jobId; PK eventTime; PK sequence; elapsedSeconds; tablesUsed
SqlSqldml: PK,FK1 jobId; PK eventTime; PK sequence; elapsedSeconds; tablesUsed; type
MemorysourceGet: PK,FK1 jobId; PK eventTime; PK sequence; cacheSize; elapsedDictSeconds; elapsedSeconds; hit

Fig. 4. Tables for monitoring

This layer of tables captures events for all D0 dbsmon databases. A brief description of these tables is provided:

- ConnmgrAcquire - Records data about who acquires a database connection.
- ConnmgrRelease - Records data about when a database connection is releaed.
- FilesourceGet - Records data about when data is requested from the file cache.
- DatacacheSet – Records data about objects being purged from the memory cache.
- MemorysourceGet – Records data about access to the memory cache.
- FilecachePrune – Records data about when files are removed from the file cache.
- ProxysourceGet – Records data about when a request is made from a remote dbserver.
- ProxysourceEstablishProxyServant – Records data about when the servant to a remote dbserver is established.
- SqlQuery – Records data about individual queries made to the underlying database.
- SqlArrayInsertLists – Records data about array inserts made to the database.
- SqlSqldml – Records data about any not query or array insert made. It also identifies the type of sql performed.
- D0omImplQuery (not shown) – Records data about each call made to D0omImpl's query method.
- D0omImplRawQuery (not shown) – Records data about each call made to D0omImpl's rawQuery method.

All tables used by the calibration dbservers exist in the first and second layer. SAM has added a few tables of its own. They are EventStorage, EventStorageSummary and MethodTiming. The first two tables capture data specific to DLSAM storing events in SAM. The third table records timing data on every call made to a SAM dbserver.

Components – Schema – database cleanup

All mysql databases are cleaned up by a cron job each night. Old data is removed and summary data may be created. The procedures currently differ between CDF and D0.

The cron job for cleaning D0's databases is called dumpControl. This script exists as part of dbsmon_col in a template form. It is intended to be copied from dbsmon_col into an account that has been setup to speak to a mysql database. Before its first usage dumpControl must be properly configured. To do this the file must be edited and the variables must be set. All required variables are identified at the top of the file. The dumpMaster script must be added to the crontab file. dumpMaster will call dumpControl.

A second script exists for loading data into the summary tables. This script is called produceSummaryData and must be added to the crontab file. It does require a parameter which identifies the name of the mysql database to summarize data over.

Plot Descriptions

Top Users

This plot lists the top 20 users who have made most connections to the server. This should not be construed to mean the clients used those connections.

Top Tables

This plot lists the top 20 tables used by the server, showing the number of access made to each table.

Query Duration

This plot shows the amount of time it takes for the server to issue queries to the database and a response returned to the server.

Latency

This scatter plot shows the time it takes for each request received from the client to be processed by the server and made ready for return to the client.
An example for the SMT server is shown in Fig. 7.

Latency Duration

This plot shows the time it taken for request received from the client to be processed by the server and made ready for return to the client.

Requests Vs Queries

This plot compares the number of client requests received each hour to the number of query requests the server makes.

Connections Per Server

This plot shows the number of connections made to each server. This should not be construed to mean the clients used those connections.

Request Per Server

This plot shows the number of requests that have been made to each server per hour.

Queries Per Server

This plot shows the number of queries that each server has made to the database per hour. An example for all the the User Calib servers is shown in Fig. 5 and SAM servers is shown in Fig. 6.

Some plots examples follows:

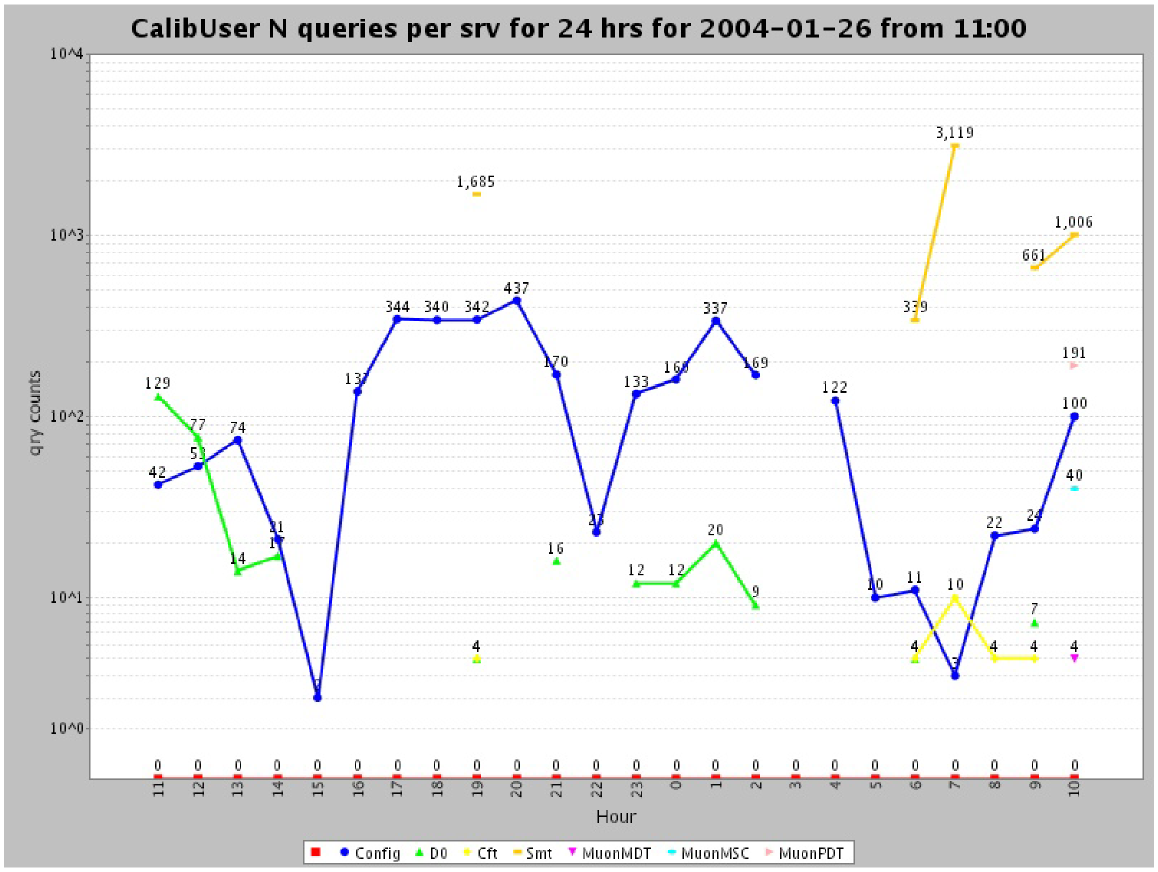


Fig. 5

The line plot in Fig. 5. shows for a period of 24 hours the number of requests for each server on an hourly base.

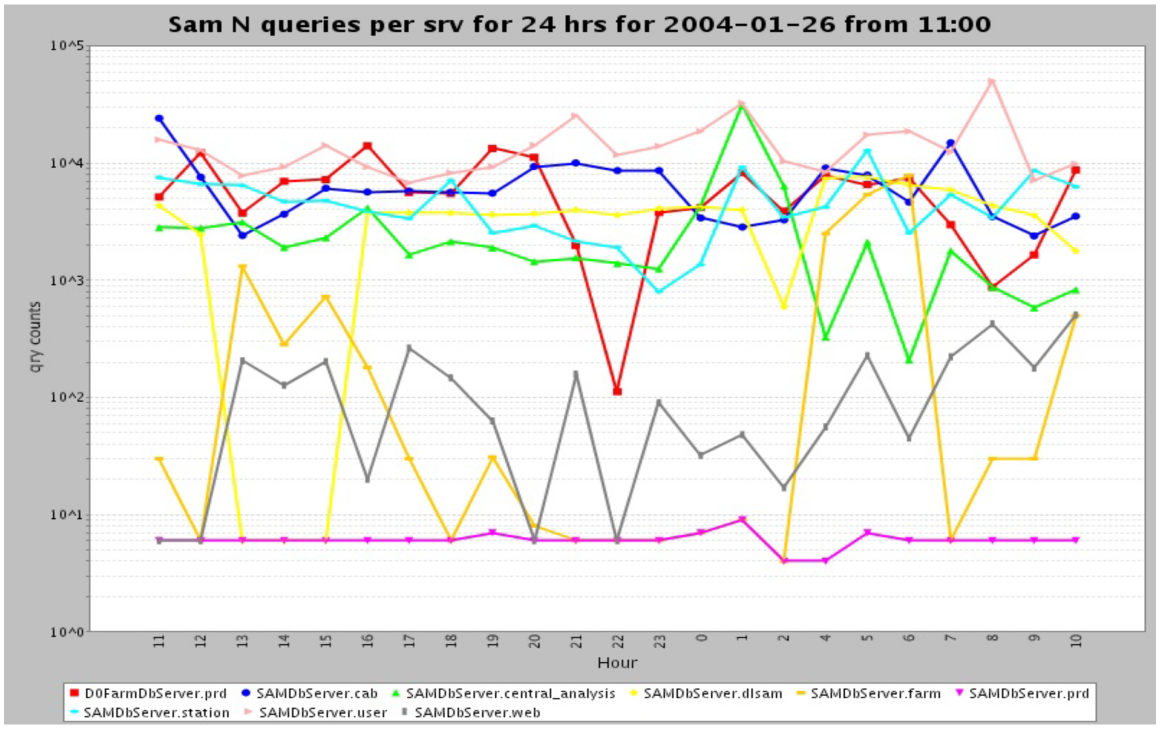


Fig. 6

Fig. 6 above gives a global behavior for 24 hours for all the SAM servers for the number of queries sent and summarized hourly.

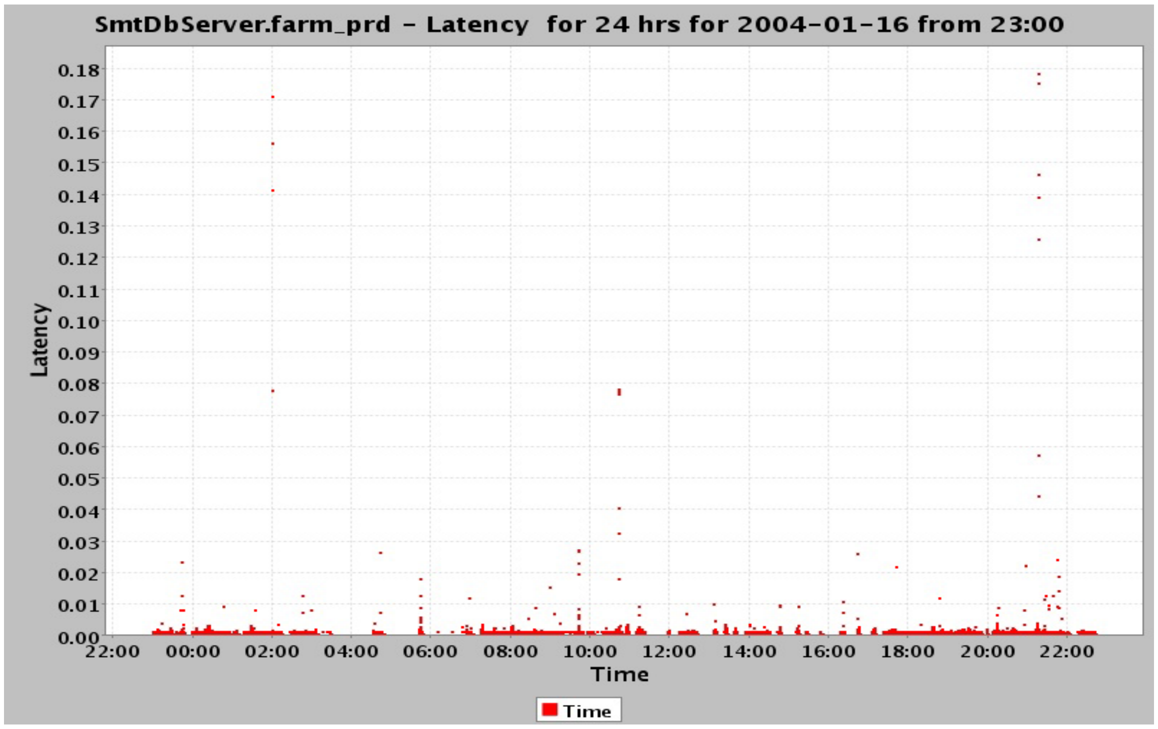


Fig. 7

Fig. 7 above shows a latency plot for the SMT Farm server .

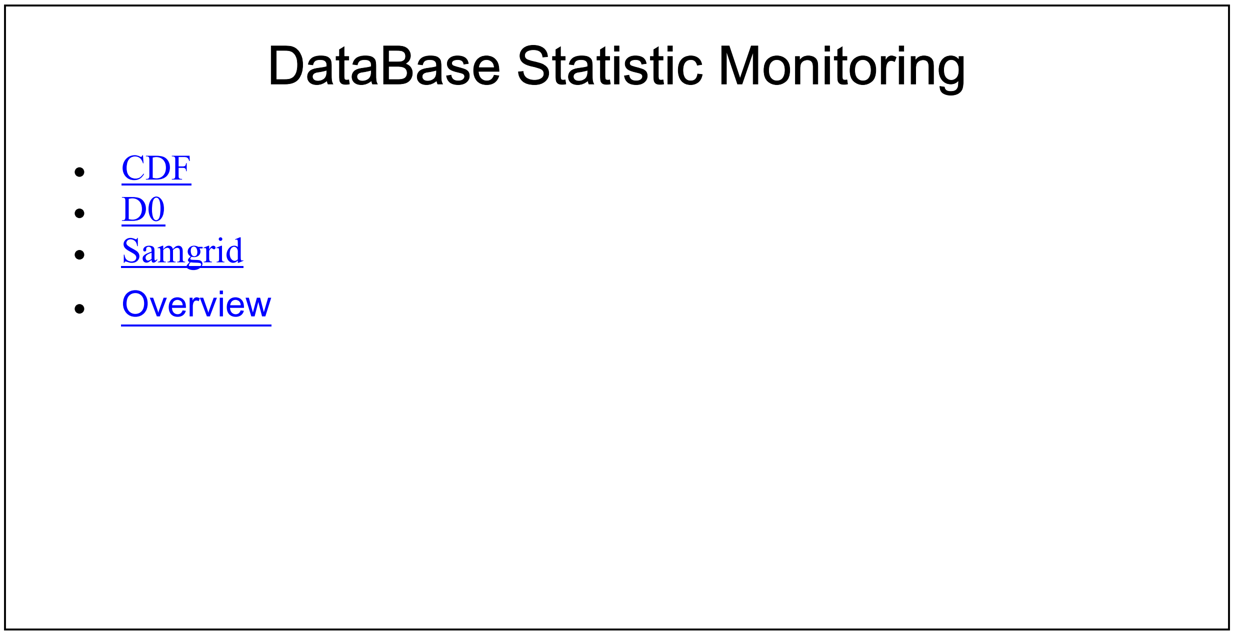
7. DBSMON Plots and WEB access

DBSMON page can be easily access at the following URL:

http://dbsmon.fnal.gov/

The user will be presented with choices to navigate the currently supported experiment and or project specific web pages:

An example of the full CDF and D0 page follows.
The full CDF and D0 pages show the plots available for viewing for the current date



DataBase Statistic Monitoring

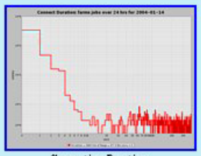
- CDF
- D0
- Samgrid
- Overview

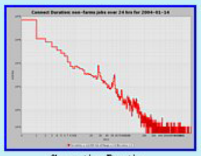
And reflect statistics on data recorded over 24 hours . Clicking on a picture, the thumbnail will show the whole picture and additional available information.

Static Plots from Database Monitoring

(All 24-Hour Plots Are from Midnight to Midnight)

Query Duration


Connection Duration
for CdfOfPrd


Connection Duration
for CdfOfRead

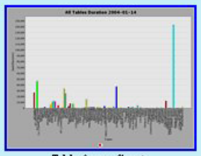
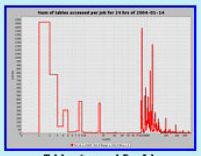


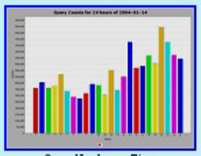
Table Access Counts

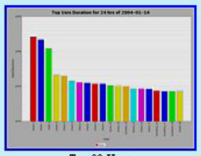
Queries per Job

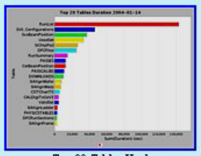
Connections per Job
for CdfOfPrd

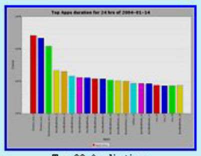
Connections per Job
for CdfOfRead

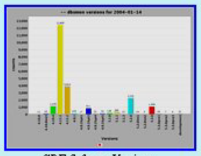

Tables Accessed Per Job

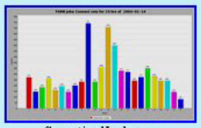

Query Number vs Time

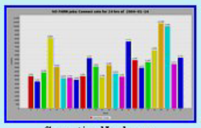

Top 20 Users

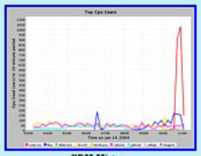

Top 20 Tables Used


Top 20 Applications


CDF Software Versions


Connection Number vs
Time for CdfOfPrd

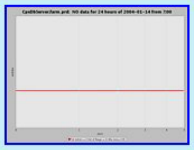
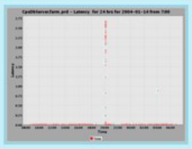
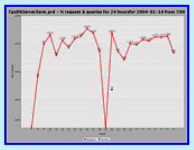
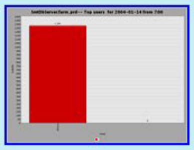
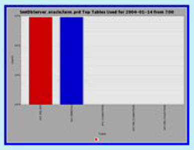
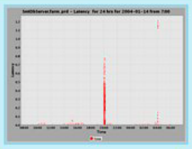
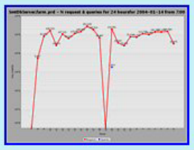
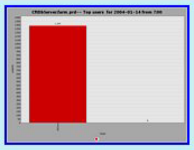
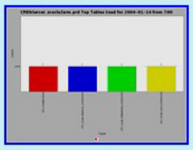
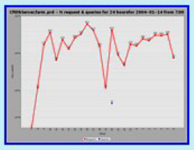
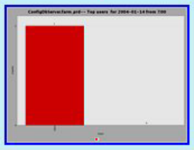
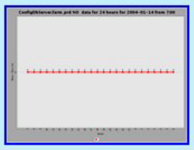
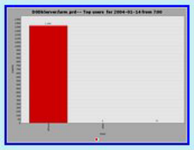
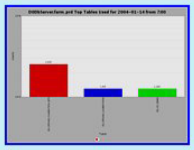
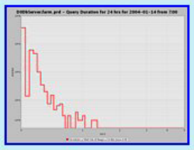
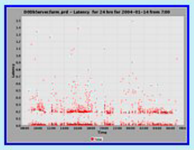
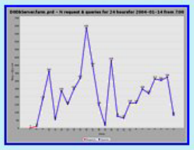

Connection Number vs
Time for CdfOfRead


CPU History

Archive
Summary

Static Plots for D0 CAlibFarm Monitoring

(All Plots Are for 24-Hour Interval)

	Top Users	Top Tables	Query Duration	Latency	Requests Vs Queries
CPS					
SMT					
CFT					
Config					
D0					

8. Future plans

8.1 Dynamic plots

So far the plots created are statically stored and viewabel from the web pages.
A future upgrade of the project will include a twofold dynamic plot generation:

- WEB pages dynamiccaly generated based on the viewer specified criteria, for example view on the same page the same type of plot let's say for a week for comparison purposes.
- The plot itself is dynamically generated sending the query. Realistically, this will probably be done using data from the summary table considering timing issues .

8.2 DBSmon extensions

An immediate re-use of dbsmon is samgrid monitoring. We are doing pro-typing now.
We import data from xml db's into a central mysql db and use dbsmon tools to plot.

8.3 CDF dbsmon extensions

Collecting information on more of the CDF tables, for example
RUNCONFIGURATIONS.